EXHIBIT D

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, vs. GOOGLE INC. Defendants.	Case No. 3:10-cv-03561-WHA

EXPERT REPORT OF DR. BENJAMIN F. GOLDBERG

REGARDING VALIDITY OF PATENTS-IN-SUIT

SUBMITTED ON BEHALF OF PLAINTIFF ORACLE AMERICA, INC.

execute the indicated action. It is inefficient, however, for interpreters to resolve the same symbolic references repeatedly. As James Gosling, the inventor of the ’104 patent, explained, “each time an instruction comprising a symbolic reference is interpreted, execution is slowed significantly.” (’104, 2:13-15.) Accordingly, there was a long-felt need to increase the speed at which interpreters executed code containing symbolic references.

421. The ’104 patent satisfied this need by designing an interpreter that operated on intermediate form object code and, whenever it resolves a symbolic reference to data, stores the corresponding numerical (*i.e.*, memory location) reference for later use. (*See generally* ’104 patent.) When the interpreter described in the patent encounters a subsequent reference to the data, it simply goes to the corresponding memory location rather than performing another time-consuming symbolic reference resolution. (*See, e.g.*, *id.* at 2:35-59.) The ’104 patent thus eliminated the need to resolve the same symbolic reference twice. (*See, e.g.*, ’104, 2:60-67.) As summarized in the ’104 patent:

> As a result, the ‘compiled’ intermediate form object code of a program achieves execution performance substantially similar to that of the traditional compiled object code, and yet it has the flexibility of not having to be recompiled when the data objects it deals with are altered like that of the traditional translated code, since data reference resolution is performed at the first execution of a generated instruction comprising a data reference. (*Id.* at 2:60-67.)

422. The ’104 patent reduced the number of symbolic reference resolutions that occur during run time and thus solved the need to quickly execute intermediate form object code having symbolic references.

2. The ’104 Patent Led to Commercial Success

423. I understand that Sun Microsystems and Oracle have implemented the claimed invention of the ’104 patent in their Java virtual machines. In May 1996, James Gosling and Henry McGilton co-authored a white paper entitled “The Java Language Environment,” in which they describe symbolic reference resolution for Java. (James Gosling & Henry McGilton, *White*

Paper, The Java Language Environment (May 1996), *available at* http://java.sun.com/docs/white/langenv/.) The white paper documents the core pieces of Java, including symbolic reference resolution as disclosed in the '104 patent.

424. The white paper explains, "Java's memory management model is based on *objects* and *references* to objects." (*Id.* at ch.2.1.6 (emphases in original).) Java bytecode references objects via symbolic references "that are resolved to real memory addresses at run time by the Java interpreter." (*Id.* at ch.6.1.) The chapter on "Interpreted and Dynamic" further explains symbolic reference resolution:

> The Java compiler doesn't compile references down to numeric values—instead, it passes symbolic reference information through to the byte code verifier and the interpreter. The Java interpreter performs final name resolution once, when classes are being linked. Once the name is resolved, the reference is rewritten as a numeric offset, enabling the Java interpreter to run at full speed. (*Id.* at ch.5.1.2.)

425. Therefore, Java interpreter only needs to incur "the small expense of a name lookup the first time any name is encountered" and need not incur the expense the second time that name is encountered. (*Id.*) After the interpreter performs the first name lookup, it can simply reference the "numeric offset." (*Id.*) In this way, the '104 patent allows "the Java interpreter to run at full speed." (*Id.*)

426. Others in the field have recognized Java's execution performance. (*See, e.g.*, Patrick Niemeyer & Joshua Peck, *Exploring Java*, Ch. 1.2 (O'Reilly 2d Ed. 1997), *available at* http://doc.novsu.ac.ru/oreilly/java/exp/index.htm (Although "[i]n general, interpreters are slow . . . Java is a fast interpreted language.").) Some consider Java as "a top performer along with C++ in many cases" even though Java requires an extra step of interpretation. (Carmine Mangione, *Performance tests show Java as fast as C++*, JavaWorld (Feb. 1, 1998), *available at* http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf.html.)

427. I understand that testimony at trial will show customer demand for devices with faster execution performance. Because the '104 patent increases Java interpreters' execution

speed, it has contributed to Java's acceptance in the market as a fast interpreted language. Therefore, I understand that there is a nexus between the claimed invention of the '104 patent and Java's commercial success.

428. Similarly, the '104 patent helps Android achieve good execution performance. I have read Professor Mitchell's Opening Patent Infringement Report, Section VI entitled "RE38,104 (Reference Resolution)" and understand that the evidence at trial will show that Android's Dalvik VM and the dexopt tool that optimizes .dex files both employ the '104 patent. Specifically, I understand that the evidence at trial will show that Dalvik VM and dexopt replace symbolic references with numeric references such as a simple integer v-table offset. Google has characterized the symbolic reference resolution as an "optimization" and has featured it in a presentation describing the implementation of the Dalvik virtual machine. (Google I/O Android Video on "Dalvik Virtual Machine Internals" by Dan Bornstein (2008), *available at* http://developer.android.com/videos/index.html#v=ptjedOZEXPM.) Therefore, Google also acknowledges how symbolic reference resolution increases execution speed and has marketed its significance through a Google I/O presentation to software developers.

429. Furthermore, I have read Professor Mitchell's Opening Patent Infringement Report, Section IV B, entitled "The Claimed Invention in the Patents-in-Suit are Necessary to Achieve Sufficient Performance and Security". I understand that Dr. Mitchell, in consultation with Oracle Java engineers Bob Vandette and Dr. Peter B. Kessler, have conducted benchmark testing and analysis of the technology described in the '104 patent, and they have confirmed that the performance of Android would be poor without the benefit of using the '104 patent. I further understand that the performance benchmark testing results show an execution speed improvement of as much as 13 times with the '104 patent than without the '104 patent.

430. I understand that testimony at trial will show customer demand for devices with faster execution performance. Based on the benchmark analysis, I conclude that Android would have been a slower, and thus less attractive platform if it had not implemented the '104 patent.

Therefore, I understand that there is a nexus between the claimed invention of the ’104 patent and Android’s commercial success.

431. For at least the above reasons, it is my opinion that secondary considerations demonstrate the non-obviousness of the ’104 patent.

B. ’205 patent

1. The ’205 Patent Solved a Long-Felt Need

432. Traditional Just-In-Time (“JIT”) Java compilers translate Java bytecode into native machine code continuously during runtime, compiling the bytecode “just-in-time” before it is about to be loaded or executed. Traditional JIT compilers then cache the compiled code for later use. Symantec Corporation’s JIT compiler, which Sun licensed and integrated into JDK 1.1, is an example of such a traditional JIT compiler. (*See* Symantec’s Just-In-Time Java Compiler to be Integrated Into Sun JDK 1.1 (Apr. 7, 1997), *available at* http://www.symantec.com/about/news/release/article.jsp?prid=19970407_03 (Symantec’s JIT compiler “instantly convert[s] Java bytecode to native code on the fly . . . .”).)

433. With JIT compilation, “[O]verall program execution time now includes JIT compilation time, in contrast to the traditional methodology of performance measurement, in which compilation time is ignored.” (Ali-Reza Adl-Tabatabai et al., *Fast, Effective Code Generation in a Just-In-Time Java Compiler*, 33 PLDI ’98 Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design & Implementation, 280, 280 (1998).) “As a result, it is extremely important for the compiler optimizations to be lightweight and effective.” (*Id.*) Furthermore, “native code generated by a JIT compiler does not always run faster than code executed by an interpreter. For example, if the interpreter is not spending the majority of its time decoding the Java virtual machine instructions, then compiling the instructions with a JIT compiler may not increase the execution speed.” (’205, 2:5-10.) Accordingly, as the ’205 patent inventors recognized, “there [was] a need for new techniques for increasing the execution speed of computer programs that are being interpreted.” (*Id.* at 2:27-29.) “Additionally, there [was] a

Dated: August 25, 2011

[illegible]

DR. BENJAMIN F. GOLDBERG